PROJECT AUTOMATA DAN TEKNIK KOMPILASI

KELOMPOK DESAIN SIMPLENEURAL-DSL

(*Dosen : Sherly Gina Supratman, M.Kom.*)

Disusun oleh :

1. Asep Haryana Saputra (20230810043)
2. Muhammad Rizal Nurfirdaus (20230810088)
3. Rio Andika Andriansyah (20230810155)

Kelas : TINFC-2023-04

TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS KUNINGAN

2026

KATA PENGANTAR

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, makalah yang berjudul **"Desain SimpleNeural-DSL"** ini dapat disusun dan diselesaikan dengan baik. Makalah ini disusun sebagai salah satu bentuk pemenuhan tugas akademik pada mata kuliah **Automata dan Teknik Kompilasi**, sekaligus sebagai upaya untuk memperdalam pemahaman mengenai konsep perancangan bahasa domain spesifik (*Domain-Specific Language/DSL*) serta kaitannya dengan teori automata dan proses kompilasi.

Makalah ini membahas konsep dasar **SimpleNeural-DSL**, yang meliputi latar belakang pengembangan DSL, perancangan sintaks dan semantik sederhana, pemodelan struktur bahasa menggunakan konsep automata, serta tahapan dasar dalam teknik kompilasi seperti analisis leksikal, analisis sintaksis, dan pemetaan ke representasi internal. SimpleNeural-DSL dirancang sebagai bahasa sederhana untuk merepresentasikan model jaringan saraf tiruan secara ringkas dan terstruktur, sehingga memudahkan pengguna dalam mendeskripsikan komponen neural network tanpa harus berinteraksi langsung dengan bahasa pemrograman tingkat rendah.

Dalam konteks mata kuliah **Automata dan Teknik Kompilasi**, proyek ini memiliki relevansi yang kuat karena mengintegrasikan teori automata sebagai dasar pemodelan bahasa, serta prinsip kompilasi sebagai mekanisme penerjemahan instruksi dari bahasa tingkat tinggi ke bentuk yang dapat diproses oleh sistem. Dengan demikian, proyek ini diharapkan dapat memberikan gambaran nyata mengenai penerapan teori yang dipelajari dalam pengembangan sistem komputasi modern.

Penulis menyadari bahwa dalam penyusunan makalah ini masih terdapat keterbatasan, baik dari segi kelengkapan materi maupun kedalaman pembahasan. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang bersifat membangun dari pembaca guna penyempurnaan makalah ini di masa yang akan datang.

Akhir kata, penulis mengucapkan terima kasih kepada dosen pengampu mata kuliah **Automata dan Teknik Kompilasi** serta semua pihak yang telah membantu dalam penyusunan makalah dan proyek ini. Semoga makalah ini dapat memberikan manfaat dan menambah wawasan bagi pembaca dalam memahami perancangan bahasa pemrograman sederhana serta penerapan automata dan teknik kompilasi dalam pengembangan **SimpleNeural-DSL**.

Kuningan, 19 Januari 2026

Kelompok DSL

Daftar Isi

BAB I
PENDAHULUAN

1. Latar Belakang & Tema Proyek

Perkembangan ilmu komputer modern menunjukkan pergeseran paradigma yang signifikan dari pemrograman imperatif tingkat rendah menuju abstraksi yang lebih tinggi dan spesifik terhadap domain permasalahan. Kompleksitas sistem komputasi, khususnya dalam bidang *Machine Learning* dan *Artificial Intelligence*, menuntut mekanisme yang mampu menyederhanakan proses perancangan model tanpa mengorbankan fleksibilitas dan ketepatan semantik. Dalam konteks ini, *Domain Specific Language* (DSL) muncul sebagai solusi konseptual dan teknis untuk menjembatani kebutuhan ekspresivitas domain dengan efisiensi implementasi sistem.

Di balik kemudahan penggunaan sebuah bahasa pemrograman, tersembunyi fondasi teoritis yang kuat berupa **teori automata, bahasa formal, dan teknik kompilasi**. Setiap pernyataan program, sesederhana apa pun tampilannya, pada hakikatnya merupakan hasil dari proses transformasi formal yang melibatkan analisis leksikal, analisis sintaksis, analisis semantik, hingga pembangkitan kode. Mata kuliah **Automata dan Teknik Kompilasi** mempelajari prinsip-prinsip fundamental ini sebagai dasar pemahaman bagaimana bahasa pemrograman dirancang, dianalisis, dan direalisasikan secara sistematis.

Proyek **DESAIN SIMPLENEURAL-DSL** dikembangkan dalam kerangka pemikiran tersebut, dengan tujuan menghadirkan sebuah DSL deklaratif yang memungkinkan pengguna mendeskripsikan konfigurasi *neural network* secara ringkas, terstruktur, dan bebas dari kompleksitas teknis bahasa pemrograman umum. SimpleNeural-DSL dirancang untuk mendefinisikan dataset, arsitektur model, optimizer, serta parameter pelatihan, yang selanjutnya dikompilasi secara otomatis menjadi kode Python berbasis TensorFlow/Keras yang dapat dieksekusi.

Dari perspektif teoritis, proyek ini merepresentasikan penerapan langsung konsep **automata hingga kompilator** dalam sebuah sistem nyata. Proses *lexical analysis* diimplementasikan menggunakan *Deterministic Finite Automaton* (DFA) berbasis *regular expression* untuk mengenali token-token DSL. Selanjutnya, *syntax analysis* dilakukan melalui *Recursive Descent Parsing* dengan pendekatan LL(1) untuk membangun *Abstract Syntax Tree* (AST) yang merepresentasikan struktur gramatikal program. Tahapan ini dilengkapi dengan *semantic analysis* untuk memastikan konsistensi tipe, validitas parameter, serta integritas simbol, sebelum akhirnya kode sumber Python dihasilkan melalui *code generation*.

Dengan mengintegrasikan teori automata, tata bahasa bebas konteks, dan pipeline kompilasi klasik ke dalam domain *machine learning*, proyek ini menunjukkan bahwa konsep-konsep formal yang dipelajari dalam Automata dan Teknik Kompilasi bukan sekadar abstraksi matematis, melainkan fondasi praktis dalam pembangunan sistem perangkat lunak modern. Oleh karena itu, pendahuluan ini menjadi landasan konseptual bagi pembahasan lebih lanjut

mengenai desain bahasa, spesifikasi token dan grammar, perancangan parser, struktur AST, hingga simulasi automata yang digunakan dalam implementasi SimpleNeural-DSL.

1.1 Tema Proyek

Proyek ini mengimplementasikan sebuah Domain Specific Language (DSL) bernama SimpleNeural-DSL untuk konfigurasi model Machine Learning. DSL ini memungkinkan pengguna mendefinisikan arsitektur neural network, konfigurasi dataset, optimizer, dan parameter training menggunakan sintaks yang sederhana dan deklaratif, yang kemudian dikompilasi menjadi kode Python dengan TensorFlow/Keras.

1.2 Kasus yang Dikerjakan

Sistem compiler ini menangani proses transformasi dari kode DSL menjadi kode Python yang executable, dengan komponen utama:

- Lexer (Tokenizer): Menganalisis input dan mengidentifikasi token menggunakan Regular Expression dan DFA
- Parser: Membangun Abstract Syntax Tree (AST) menggunakan Recursive Descent Parsing (LL(1))
- Semantic Analyzer: Melakukan type checking, validasi parameter, dan symbol table management
- Code Generator: Menghasilkan kode Python yang menggunakan TensorFlow/Keras

1.3 Contoh Input DSL

```
# SimpleNeural DSL - Klasifikasi Iris
DATASET load "Iris.csv" TARGET "Species"

MODEL "IrisClassifier" {
    LAYER DENSE units: 64 activation: "relu"
    LAYER BATCHNORM
    LAYER DENSE units: 32 activation: "relu"
    LAYER DROPOUT rate: 0.2
    LAYER DENSE units: 3 activation: "softmax"

    OPTIMIZER "adam" lr: 0.01
    TRAIN epochs: 50 batch_size: 16 validation_split: 0.15
}
```

BAB II
ANALISIS LEKSIKAL (LEXER)

2. Daftar Token, Regular Expression, dan Sketsa NFA/DFA

2.1 Daftar Token

Token Type	Description	Pattern (Regex)	Example
KEYWORD_DATASET	Deklarasi dataset	\bDATASET\b	DATASET
KEYWORD_MODEL	Deklarasi model	\bMODEL\b	MODEL
KEYWORD_LAYER	Deklarasi layer	\bLAYER\b	LAYER
KEYWORD_DENSE	Layer type Dense	\bDENSE\b	DENSE
KEYWORD_CONV2D	Layer type Conv2D	\bCONV2D\b	CONV2D
KEYWORD_DROPOUT	Layer type Dropout	\bDROPOUT\b	DROPOUT
KEYWORD_FLATTEN	Layer type Flatten	\bFLATTEN\b	FLATTEN
KEYWORD_LSTM	Layer type LSTM	\bLSTM\b	LSTM
KEYWORD_GRU	Layer type GRU	\bGRU\b	GRU
KEYWORD_BATCHNORM	Batch Normalization	\bBATCHNORM\b	BATCHNORM
KEYWORD_MAXPOOL2D	Max Pooling 2D	\bMAXPOOL2D\b	MAXPOOL2D
KEYWORD_OPTIMIZER	Konfigurasi optimizer	\bOPTIMIZER\b	OPTIMIZER
KEYWORD_TRAIN	Konfigurasi training	\bTRAIN\b	TRAIN
KEYWORD_LOAD	Load dataset	\bload\b	load
KEYWORD_TARGET	Target column	\bTARGET\b	TARGET
KEYWORD_UNITS	Parameter units	\bunits\b	units
KEYWORD_ACTIVATION	Parameter activation	\bactivation\b	activation
KEYWORD_LR	Learning rate	\blr\b	lr
KEYWORD_EPOCHS	Parameter epochs	\bepochs\b	epochs
KEYWORD_BATCH_SIZE	Parameter batch size	\bbatch_size\b	batch_size
STRING	String literal	"[^"]*"	"data.csv"
INTEGER	Integer literal	\d+	128
FLOAT	Float literal	\d+\.\d+	0.01
BOOLEAN	Boolean literal	\b(true|false)\b	true
TUPLE	Tuple literal	\(\s*\d+\s*,\s*\d+\s*\)	(3,3)
IDENTIFIER	Nama variabel	[a-zA-Z_][a-zA-Z0-9_]*	model_satu
LBRACE	Left brace	\{	{
RBRACE	Right brace	\}	}

COLON	Colon	:	:
COMMA	Comma	,	,
NEWLINE	Line break	\n	-
COMMENT	Comment (ignored)	#.*	# komentar
EOF	End of file	-	-

2.2 DFA untuk Number Recognition

DFA (Deterministic Finite Automaton) untuk mengenali token INTEGER dan FLOAT:

States: Q = {q0, q1, q2, q3}
Start State: q0
Final States: F = {q1, q3}
Alphabet: Σ = {0-9, .}

Transisi:
• q0 --[digit 0-9]--> q1 (mulai angka)
• q1 --[digit 0-9]--> q1 (lanjut digit integer)
• q1 --[dot .]--> q2 (titik desimal)
• q2 --[digit 0-9]--> q3 (bagian fractional)
• q3 --[digit 0-9]--> q3 (lanjut fractional)
• q1 --[other]--> ACCEPT INTEGER
• q3 --[other]--> ACCEPT FLOAT

State Transition Table untuk Number:

State	digit [0-9]	dot [.]	other
q0	q1	ERROR	ERROR
q1	q1	q2	ACCEPT (INTEGER)
q2	q3	ERROR	ERROR
q3	q3	ERROR	ACCEPT (FLOAT)

2.3 DFA untuk String Recognition

States: Q = {q0, q1, q2}
Start State: q0
Final States: F = {q2}
Alphabet: Σ = {", any char}

Transisi:
• q0 --[quote "]--> q1 (mulai string)
• q1 --[any char except "]--> q1 (karakter dalam string)
• q1 --[quote "]--> q2 (akhir string - ACCEPT STRING)

2.4 DFA untuk Identifier/Keyword Recognition

States: Q = {q0, q1, q2}
Start State: q0
Final States: F = {q2}
Alphabet: Σ = {a-z, A-Z, 0-9, _}

Transisi:
• q0 --[letter/underscore a-zA-Z_]--> q1 (mulai identifier)
• q1 --[letter/digit/underscore a-zA-Z0-9_]--> q1 (lanjut identifier)
• q1 --[other]--> q2 (cek keyword table)

Catatan: Setelah mencapai q2, sistem melakukan pengecekan pada keyword table. Jika token ditemukan dalam keyword table maka dikembalikan sebagai KEYWORD, jika tidak maka dikembalikan sebagai IDENTIFIER.

BAB III
ANALISIS SINTAKSIS (GRAMMAR)

3. Context-Free Grammar (CFG) yang Digunakan

3.1 Grammar Rules (BNF Notation)

```
<program>           ::= <dataset_decl>? <model_decl>+

<dataset_decl>      ::= DATASET LOAD STRING TARGET STRING

<model_decl>        ::= MODEL STRING LBRACE <model_body> RBRACE

<model_body>        ::= <layer_decl>+ <optimizer_decl>? <train_decl>?

<layer_decl>        ::= LAYER <layer_type> <layer_params>

<layer_type>        ::= DENSE | CONV2D | DROPOUT | FLATTEN | LSTM | GRU |
BATCHNORM | MAXPOOL2D

<layer_params>      ::= <param_item>*

<param_item>        ::= PARAM_NAME COLON <value>

<optimizer_decl>    ::= OPTIMIZER STRING <optimizer_params>?

<optimizer_params>  ::= <param_item>*

<train_decl>        ::= TRAIN <train_params>

<train_params>      ::= <param_item>+

<value>             ::= STRING | INTEGER | FLOAT | BOOLEAN | TUPLE |
IDENTIFIER
```

3.2 FIRST dan FOLLOW Sets

FIRST Sets:

Non-Terminal	FIRST Set
program	{ DATASET, MODEL }
dataset_decl	{ DATASET }
model_decl	{ MODEL }
layer_decl	{ LAYER }
layer_type	{ DENSE, CONV2D, DROPOUT, FLATTEN, LSTM, GRU, BATCHNORM, MAXPOOL2D }
optimizer_decl	{ OPTIMIZER }
train_decl	{ TRAIN }
param_item	{ units, activation, lr, epochs, batch_size, ... }

value	{ STRING, INTEGER, FLOAT, BOOLEAN, TUPLE, IDENTIFIER }

FOLLOW Sets:

Non-Terminal	FOLLOW Set
program	{ EOF }
dataset_decl	{ MODEL }
model_decl	{ MODEL, EOF }
layer_decl	{ LAYER, OPTIMIZER, TRAIN, RBRACE }
layer_type	{ IDENTIFIER, LAYER, OPTIMIZER, TRAIN, RBRACE }
optimizer_decl	{ TRAIN, RBRACE }
train_decl	{ RBRACE }
param_item	{ IDENTIFIER, LAYER, OPTIMIZER, TRAIN, RBRACE, NEWLINE }
value	{ IDENTIFIER, LAYER, OPTIMIZER, TRAIN, RBRACE, NEWLINE }

BAB IV
PERANCANGAN PARSER DAN AST

4. Desain Parser (Recursive Descent/LL(1)) dan Sketsa AST

4.1 Metode Parsing: Recursive Descent

Parser menggunakan teknik Recursive Descent Parsing yang merupakan implementasi dari LL(1) parser. Setiap non-terminal dalam grammar diimplementasikan sebagai fungsi yang memanggil fungsi lain secara rekursif sesuai dengan production rules.

Fungsi-fungsi Parser Utama:

Fungsi Parser	Non-Terminal	Deskripsi
parse_program()	<program>	Entry point, parse dataset opsional dan model wajib
parse_dataset()	<dataset_decl>	Parse deklarasi DATASET load ... TARGET ...
parse_model()	<model_decl>	Parse deklarasi MODEL dengan body dalam {}
parse_layer()	<layer_decl>	Parse definisi layer dengan tipe dan parameter
parse_optimizer()	<optimizer_decl>	Parse konfigurasi optimizer
parse_train_config()	<train_decl>	Parse konfigurasi training
parse_parameter()	<param_item>	Parse parameter dalam format nama: nilai

4.2 Struktur Abstract Syntax Tree (AST)

AST direpresentasikan menggunakan node-node hierarkis dengan tipe-tipe berikut:

Node Type	Atribut	Deskripsi
ProgramNode	dataset, models[]	Root node, berisi dataset dan daftar model
DatasetNode	file_path, target_column	Informasi dataset yang di-load
ModelNode	name, layers[], optimizer, train_config	Definisi model neural network
LayerNode	layer_type, parameters{}	Satu layer dengan tipe dan parameter
OptimizerNode	optimizer_type, parameters{}	Konfigurasi optimizer
TrainConfigNode	parameters{}	Konfigurasi training (epochs, batch_size, dll)
ParameterNode	name, value	Pasangan parameter key-value

4.3 Sketsa AST untuk Contoh Input

Untuk input DSL pada bagian 1.3, AST yang dihasilkan adalah:

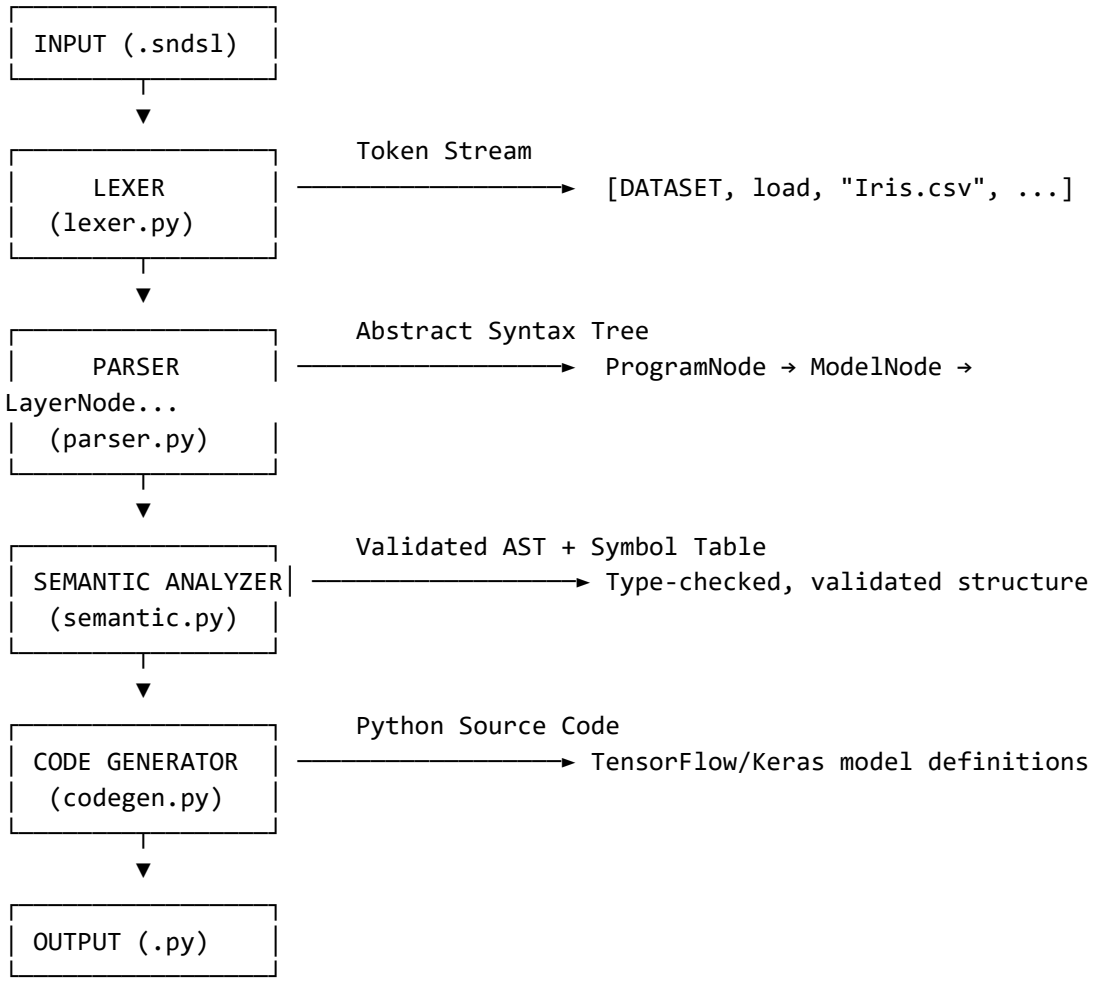
```
ProgramNode
├── DatasetNode
│   ├── file_path: "Iris.csv"
│   └── target_column: "Species"
└── ModelNode: "IrisClassifier"
    ├── LayerNode[0]: DENSE
    │   ├── units: 64
    │   └── activation: "relu"
    ├── LayerNode[1]: BATCHNORM
    ├── LayerNode[2]: DENSE
    │   ├── units: 32
    │   └── activation: "relu"
    ├── LayerNode[3]: DROPOUT
    │   └── rate: 0.2
    ├── LayerNode[4]: DENSE
    │   ├── units: 3
    │   └── activation: "softmax"
    ├── OptimizerNode: "adam"
    │   └── lr: 0.01
    └── TrainConfigNode
        ├── epochs: 50
        ├── batch_size: 16
        └── validation_split: 0.15
```

BAB V
ANALISIS SEMANTIK DAN CODE GENERATION

5. Desain IR/DSL dan Alur Eksekusi

5.1 Pipeline Kompilasi

Sistem SimpleNeural-DSL mengikuti pipeline compiler tradisional dengan tahapan sebagai berikut:



5.2 Semantic Analysis dan validasi

Semantic Analyzer melakukan validasi terhadap beberapa aspek:

Parameter	Tipe	Validasi
units	Integer	value > 0
activation	String	value ∈ {relu, sigmoid, tanh, softmax, linear, selu, elu, swish, gelu}
lr (learning_rate)	Float	0.0001 ≤ value ≤ 1.0

epochs	Integer	value > 0
batch_size	Integer	value > 0
validation_split	Float	0.0 ≤ value < 1.0
rate (dropout)	Float	0.0 ≤ value ≤ 1.0
optimizer_type	String	value ∈ {adam, sgd, rmsprop, adagrad, adamw, nadam}
layer_type	String	value ∈ {DENSE, CONV2D, DROPOUT, FLATTEN, LSTM, GRU, BATCHNORM, MAXPOOL2D}

5.3 Code Generation

Code Generator menghasilkan kode Python yang executable dengan komponen-komponen berikut:

- Import statements (TensorFlow, Pandas, NumPy, Scikit-learn)
- Data loading dan preprocessing function
- Model building function dengan tf.keras.Sequential
- Compile function dengan loss dan metrics yang sesuai
- Training function dengan callbacks (EarlyStopping, ReduceLROnPlateau, ModelCheckpoint)
- Evaluation function dengan metrics dan classification report
- Main function untuk eksekusi keseluruhan pipeline

BAB VI
SIMULASI DAN PENGUJIAN

6. Penjelasan Simulasi Automata (DFA/PDA) dan Contoh Input-Output

6.1 Simulasi DFA pada Lexer

Lexer menggunakan DFA untuk mengenali token. Berikut simulasi untuk input "0.01":

Step	Input Char	Current State	Action
1	0	q0 → q1	Digit detected, enter integer state
2	.	q1 → q2	Dot detected, enter decimal state
3	0	q2 → q3	Digit detected, enter fractional state
4	1	q3 → q3	Continue fractional
5	EOF	q3 → ACCEPT	Return FLOAT token with value 0.01

6.2 Simulasi PDA pada Parser

Parser menggunakan implicit stack dalam recursive descent untuk mengelola konteks parsing. Simulasi untuk parsing MODEL block:

Step	Token	Stack (Implicit)	Action
1	MODEL	[parse_model]	Call parse_model(), expect MODEL
2	STRING "IrisClassifier"	[parse_model]	Store model name
3	LBRACE {	[parse_model, model_body]	Push: enter model body context
4	LAYER	[parse_model, model_body, parse_layer]	Push: call parse_layer()
5	DENSE	[parse_model, model_body, parse_layer]	Set layer_type = DENSE
6	units: 64	[parse_model, model_body, parse_layer]	Parse parameter
7	NEWLINE	[parse_model, model_body]	Pop: return from parse_layer()
8	...	...	(continue for each layer)

BAB VII
PENUTUP

7.1 Kesimpulan

Berdasarkan pembahasan yang telah diuraikan, dapat disimpulkan bahwa perancangan dan implementasi **SimpleNeural-DSL** berhasil menunjukkan bagaimana konsep-konsep fundamental dalam **Automata dan Teknik Kompilasi** dapat diterapkan secara nyata dalam pengembangan sebuah *Domain Specific Language* untuk konfigurasi model *Machine Learning*. Melalui pendekatan deklaratif, SimpleNeural-DSL mampu menyederhanakan proses pendefinisian arsitektur *neural network*, konfigurasi dataset, optimizer, dan parameter pelatihan, tanpa menghilangkan ketelitian semantik yang diperlukan dalam sistem komputasi modern.

Implementasi sistem kompilator SimpleNeural-DSL mengikuti pipeline kompilasi klasik yang terstruktur, dimulai dari tahap *lexical analysis* menggunakan *Deterministic Finite Automaton* (DFA) berbasis *regular expression* untuk pengenalan token, dilanjutkan dengan *syntax analysis* melalui *Recursive Descent Parsing* berpendekatan LL(1) untuk membangun *Abstract Syntax Tree* (AST), serta *semantic analysis* untuk melakukan validasi tipe, konsistensi parameter, dan pengelolaan simbol. Tahapan akhir berupa *code generation* berhasil menerjemahkan struktur internal DSL menjadi kode Python berbasis TensorFlow/Keras yang bersifat *executable*, sehingga membuktikan bahwa spesifikasi bahasa formal dapat diubah menjadi sistem perangkat lunak yang fungsional.

Selain itu, simulasi automata yang digunakan dalam proses lexer dan parser memperlihatkan keterkaitan langsung antara teori automata hingga *pushdown automata* dengan implementasi praktis dalam compiler modern. Hal ini menegaskan bahwa teori bahasa formal dan automata bukan sekadar konsep matematis abstrak, melainkan fondasi esensial dalam pembangunan bahasa pemrograman dan sistem penerjemah kode. Dengan demikian, proyek ini tidak hanya memenuhi tujuan akademik mata kuliah **Automata dan Teknik Kompilasi**, tetapi juga memberikan pemahaman komprehensif mengenai integrasi teori dan praktik dalam desain bahasa pemrograman berbasis domain.